STUDYDADDY

Get Homework Help From Expert Tutor

Get Help

CS 367 Fall 2020
Project 2: Defusing a Binary Bomb
Due: Friday, November 6, 2020 (11:59pm EST)

This is an individual assignment.

Introduction

The nefarious Dr. Evil has planted a slew of "binary bombs" on our machines. A binary bomb is a program that consists of a sequence of phases. Each phase expects you to type a particular string on `stdin`. If you type the correct string, then the phase is defused and the bomb proceeds to the next phase. Otherwise, the bomb explodes by printing `"BOOM!!!"` and then terminating. The bomb is defused when every phase has been defused.

There are too many bombs for us to deal with, so we are giving each student a bomb to defuse. Your mission, which you have no choice but to accept, is to defuse your bomb before the due date. Good luck, and welcome to the bomb squad!

Step 1: Get Your Bomb

Each student will attempt to defuse their own personalized bomb. Each bomb is a Linux binary executable file that has been compiled from a C program. While each bomb is unique, the phases follow basic patterns. To obtain your bomb, you need to be on a machine that can connect to **zeus-1.vse.gmu.edu**. For this project, you **must** connect to **zeus-1**. If you connect to zeus, it may take you to zeus-2, where you cannot be graded!

Login to **zeus-1.vse.gmu.edu** using ssh and your GMU userid/password. Remember that you need to be on the campus network, or be connecting remotely through the VPN software. The bottom line is that the technique you've been using to connect to zeus in the previous assignment should also work for this project.

To get your custom copy of the bomb, go to this link:

<div align="center">

http://zeus-1.vse.gmu.edu:15225/

</div>

It is critical that you connect to zeus-1 explicitly (and not to zeus-2). Also do NOT connect to simply zeus (doing so may direct you to either zeus-1 or zeus-2 randomly!).

This will give you a binary bomb request form for you to fill in. Enter your user name and GMU email address and hit the Submit button. The server will build your bomb and return it to your browser in a tar file called bombk.tar, where k is the unique number of your bomb.

Save the `bombk.tar` file on zeus and unpack it (`tar -xvf bombk.tar`).

This will create a directory called `./bombk` with the following files:

- `README`: Identifies the bomb and its owner.
- `bomb`: The executable binary bomb. This executable will only run on zeus-1.vse.gmu.edu
- `bomb.c`: Source file with the bomb's main function.

Important Notes:

1. Due to the heavy load on zeus-1, sometimes it may take up to 60-90 seconds before the bomb file is loaded on your browser. Please be patient. If you are appropriately connected (on campus network or via VPN) and the site seems to be down, please send a message on Piazza immediately so others can chime in or an instructor can restart the server.

2. As some students postpone working on the project to the last couple of days, zeus-1 may become slower and slower towards the deadline. To avoid the frustration, we strongly recommend that you complete as many phases as possible as early as possible.

Also, if you make any kind of mistake requesting a bomb (such as neglecting to save it), simply request another bomb. We see who checks out every single bomb, so you need to let us know your old bomb and new bomb (send an e-mail to your recitation GTA and CS 367 section instructor).

When you defuse a phase of the bomb (as described below), the program notifies the instructor. However, each time your bomb explodes on `zeus` it notifies the instructor, and you lose `0.5` points (up to a max of 10 points) in the final score for the project. So there are consequences to exploding the bomb. You must be careful!

Step 2: Defuse Your Bomb

You must do the assignment only on zeus-1 (not any other machine or zeus-2!). In fact, there is a rumor that Dr. Evil really is evil, and the bomb will always blow up if run elsewhere. There are several other tamper-proofing devices built into the bomb as well, or so we hear. Your project is to defuse your bomb.

You may only gdb, objdump, or **strings on zeus to defuse your bomb.** You may, of course, take the outputs of those tools and paste them into a notepad, excel, or other such text editor to help in your analysis, but you may not use any other tools on your executable.

Please look at the **hints section** for some tips and ideas.

Each time your bomb explodes it notifies the bomb server, and you lose 1/2 point (up to a max of 10 points) in the final score for the project. So there are consequences to exploding the bomb. You must be careful! The first four phases are worth 15 points each. Phases 5 and 6 are a little more difficult, so they are worth 20 points each. So the maximum score you can get is 100 points. (Some of you may have heard about a "secret phase"... What secret phase?)

Although phases get progressively harder to defuse, the expertise you gain as you move from phase to phase should offset this difficulty. However, the last phase will challenge even the best students, so please don't wait until the last minute to start.

The bomb ignores blank input lines. If you run your bomb with a command line argument, for example,

```
linux> ./bomb psol.txt
```

then it will read the input lines from `psol.txt` until it reaches `EOF` (end of file), and then switch over to `stdin`. We added this feature so you don't have to keep retyping the solutions to phases you have already defused.

To avoid accidentally detonating the bomb, you will need to learn how to use `gdb` to single-step through the assembly code and how to set breakpoints. You will also need to learn how to inspect both the registers and the memory states. One of the nice side-effects of doing the project is that you will get very good at using a debugger. This is a crucial skill that will pay big dividends the rest of your career.

Logistics

This is an individual project. Clarifications and corrections will be posted on the Piazza forum if needed.

Hand-In

There is no explicit hand-in. The bomb will notify your instructor automatically after you have successfully defused it on `zeus-1.vse.gmu.edu`. You can keep track of how you are doing by looking at:

http://zeus-1.vse.gmu.edu:15225/scoreboard

This web page is updated continuously to show the progress of the class. Note that this web page is only accessible from a computer connected to campus internet, or when using a VPN.

Hints (Please read this!)

There are many ways of defusing your bomb. You can examine it in great detail without ever running the program, and figure out exactly what it does. This is a useful technique, but it is not always easy to do. You can also run it under a debugger, watch what it does step by step, and use this information to defuse it. This is probably the fastest way of defusing it. You may not modify your bomb in any way to defuse it; your inputs are attempted on a fresh copy of your bomb for grading purposes so you can't circumvent it this way.

We make one request; **please do not use brute force!** You could write a program that will try every possible key to find the right one. But this is no good for several reasons:

- You lose 1/2 point (up to a max of 10 points) every time you guess incorrectly and the bomb explodes.

- Every time you guess wrong, a message is sent to the bomb server. You could very quickly saturate the network with these messages, and cause the system administrators to revoke your computer access.

- We haven't told you how long the strings are, nor have we told you what characters are in them. Even if you made the (incorrect) assumptions that they all are less than 80 characters long and only contain letters, then you will have 26^{80} guesses for each phase. This will take a very long time to run, and you will not get the answer before the assignment is due.

There are many tools which are designed to help you figure out both how programs work, and what is wrong when they don't work. Here is a list of some of the tools you may find useful in analyzing your bomb, and hints on how to use them.

- **gdb** The GNU debugger is a command line debugger tool available on virtually every platform. You can trace through a program line by line, examine memory and registers, look at both the source code and assembly code (we are not giving you the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts. Here are some tips for using gdb. To keep the bomb from blowing up every time you type in a wrong input, you'll want to learn how to set breakpoints.
    - The CS:APP Student Site at http://csapp.cs.cmu.edu/3e/students.html has a very handy single-page gdb summary.

    - For other documentation, type **help** at the gdb command prompt, or type **man gdb**, or **info gdb** at a Unix prompt. Some people also like to run gdb under **gdb-mode** in emacs.

- **objdump -t**   This will print out the bomb's symbol table. The symbol table includes the names of all functions and global variables in the bomb, the names of all the functions the bomb calls, and their addresses. You may learn something by looking at the function names!

- **objdump –d**   Use this to disassemble all of the code in the bomb. You can also just look at individual functions. Reading the assembler code can tell you how the bomb works. Although **objdump -d** gives you a lot of information, it doesn't tell you the whole story. Calls to system-level functions are displayed in a cryptic form. For example, a call to **sscanf** might appear as:

  8048c36: e8 99 fc ff ff call 80488d4 <_init+0x1a0>

  To determine that the call was to **sscanf**, you would need to disassemble within **gdb**.

- **strings**   This utility will display the printable strings in your bomb.

Looking for a particular tool? How about documentation? Don't forget, the command **man** is your friend. You can use this to look up information on system functions that the binary executable might use. In particular, **man ascii** might come in useful.

Advice
Based on our experience defusing bombs, we have the following advice:
- I print out all of the code associated with a particular phase when I start to diffuse it. As I work, I annotate this code with information as I figure out what is going on.

- As you will notice from examining **bomb.c**, the user input is sent into the phase as a single string. One of the first things the phase will do is try to 'parse' it into the form it is expecting.

- When you are trying to figure out what values are needed for a phase (particularly if the values are integers), try 'interesting' numbers like **42**, **-17**, ... first. The reason for this is that if you see one of these numbers appear in a register or in a memory location, chances are pretty good that was your input.

- Don't bother trying to step through system functions. This is generally a waste of time. If you don't know what the function does, use **man** to investigate the parameters and return information. Then examine the parameters that are sent and the return value.

- If you end up inside a function that you don't want to step through, the **gdb** command **finish** will take you to the end. A better approach is to use **nexti** to step over that call instead.

- The bomb functions typically have a generic name like **func4** or **phase2**. It has been my experience that if a function has a descriptive name, you can assume that the function behaves as expected and that they are NOT trying to trick you. Don't spend your time trying to reverse engineer functions like **read_six_numbers** or **strings_not_equal**.

- **Understanding parameter passing is critical to this assignment.** Whenever you see a reference to **%rsp** in the lines before a call, this involves placing a parameter on the stack. It is never a bad idea to figure out what the value of the parameter is. Remember some arguments are going to be ints! So, if you see **%esi** before a function call, you know the second argument is an int, or if it uses **%eax** after a call, it's working with an int return value.

- It may not be important to understand exactly what every statement is doing - the goal is to avoid the `explode_bomb` calls. I tend to step through the code until just before one of these calls and then focus on what conditions are not being met. You can also set a breakpoint at `explode_bomb` to ensure you don't accidently execute it.

Submission Information
There is no explicit submission. What you see on the scoreboard will reflect the score we can see.

Deadline
The deadline for this project is Friday, November 6, 2020 11:59 PM EST.

All your interactions with the server are recorded in a log – you can continue to work beyond this deadline without a penalty as long as you have available late tokens. Once you have used all of your available late tokens, there will be 25% ceiling penalty for each extra 24-hour period. At most two days of late work are accepted beyond the deadline (regardless of the number of late tokens that you have).

The CS 367 GTAs and UTAs can help you with the use of the authorized tools **gdb**, **objdump,** and **strings**.

You are encouraged to use the Piazza forum for your questions. However, please do not post long Assembly code sections from your binary bomb to Piazza and ask what the "logic" of that code is – it is your responsibility to figure out that logic. But of course, questions about the semantics of individual assembly instructions and the use of specific tools (e.g., gdb) are welcome.

Good luck and have fun defusing your binary bomb!